Evolutionary Algorithms **(Goal: get the "optimal" solution)**

Many AI applications have a search space that is exponentially proportional to the problem dimensions and but for the simplest of cases these problems cannot be solved using exhaustive search methods. Consequently, there is considerable interest in heuristic techniques that attempt to discover near-optimal solutions within an acceptable time.

In AI, an evolutionary algorithm (EA) is a generic population-based meta-heuristic optimization algorithm.
Simply put, EAs are computer programs that attempt to solve complex problems by mimicking the processes of Darwinian evolution. An EA uses mechanisms inspired by biological evolution / natural selection, such as inheritance, Reproduction (crossover and mutation), recombination, and selection.

EAs include the following search and optimization methods:

- Genetic Algorithms
- Evolutionary Programming
- Genetic Programming
- Evolution Strategies
- Classifier systems

Optimization design for a method is based on its components. The most important components in a EA method consist of:

- genetic representation of candidate solutions (definition of individuals)
- evaluation function (fitness function)
- population
- selection scheme
    - parent selection mechanism
    - survivor selection mechanism (replacement)
- Genetic/variation operators (crossover, mutation,…)

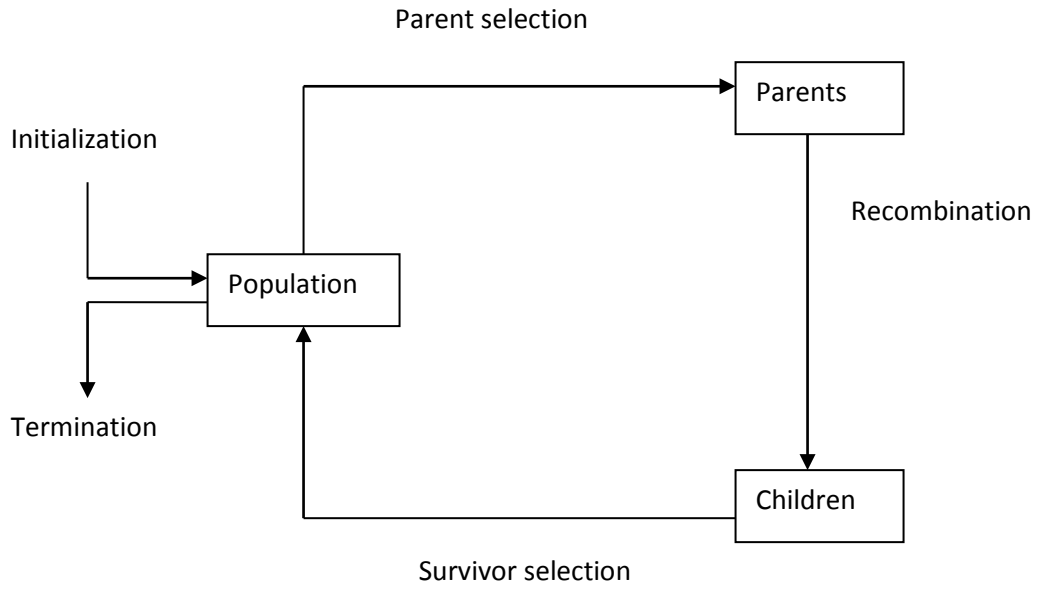
Genetic Algorithms in Search and Optimization **(Goal: achieve the global (or near global) optimum)**

Genetic algorithms (GA) are the most frequently encountered type of evolutionary algorithm.
A GA is an adaptive search heuristic that mimics the process of natural selection.

GAs provide a framework for effectively sampling large search spaces, and the basic technique is both broadly applicable and easily tailored to specific problems.
The general scheme of a GA can be given as follows:

```
begin
   INITIALIZE population with random candidate solutions;
   EVALUATE each candidate;
   repeat
      SELECT parents;
      RECOMBINE pairs of parents;
      MUTATE the resulting children;
      EVALUATE children;
      SELECT individuals for the next generation
   until TERMINATION-CONDITION is satisfied
end
```

The GA can be represented in form of a diagram:



Remember: EAs use mechanisms inspired by biological evolution, such as inheritance, Reproduction (crossover and mutation), recombination, and selection.

Biological Terminology:

- gene
    - functional entity that codes for a specific feature e.g. eye color

    - set of possible alleles
- allele
    - value of a gene e.g. blue, green, brown
    - codes for a specific variation of the gene/feature
- locus
    - position of a gene on the chromosome
- genome
    - set of all genes that define a species
    - the genome of a specific individual is called genotype
    - the genome of a living organism is composed of several chromosomes
- population
    - set of competing genomes/individuals

Reproduction methods:

There are two basic methods of reproduction:

- Mutation
- Crossover

Genetic algorithms use crossover (crossover in genetics hence the 'gene' in their name) and mutation to search the space of possible solutions. If only mutation is used, the algorithm is very slow. Crossover makes the algorithm significantly faster.

GA is a kind of **hill-climbing** search. As with all hill-climbing algorithms, there is a problem of local maxima. Local maxima in a genetic problem are those individuals that get stuck with a pretty good, but not optimal, fitness measure. Any small mutation gives worse fitness. Fortunately, crossover can help them get out of a local maximum. Also, mutation is a random process, so it is possible that we may have a sudden large mutation to get these individuals out of this situation. (In fact, these individuals never get out. It's their offspring that get out of local maxima).

One significant difference between GAs and hill-climbing is that, it is generally a good idea in GAs to fill the local maxima up with individuals. Overall, GAs have less problems with local maxima than back-propagation neural networks.

How Genetic Algorithms search for a solution / how do they work?

It is a random search that is inspired by the evolution of organisms. This generally involves:

- Parallel hill climbing with information exchange among candidate solutions
- Population of candidate solutions

- Crossover for information exchange

The details of how Genetic Algorithms work are explained below:

1. Initialization

While genetic algorithms are generally stated with an initial population that is *generated randomly*, some research has been conducted into using *special techniques* to produce a higher quality initial population. Such an approach is designed to give the GA a good start and speed up the evolutionary process.

Example: Some authors propose a GA for exam timetabling problem in which the GA works only with feasible solutions, implying that the initial population must also be made up of feasible solution. Then the GA is run to improve the fitness of the initial population.

Example: In a simple exam timetabling problem, we can use a non-binary bit string representation to represent the chromosome because it is easy to understand and represent. We use six positions representing six exams with each position's value as the time slot assigned to the exam. We can generate the population randomly to assign each exam a timeslot.

Day	AM		PM	
	time1	time2	time3	time4
Day1			e1	e3
Day2		e5	e6	e2,e4

If we randomly generate six numbers 3, 8, 4, 8, 6, 7 as six timeslots for *e1-e6*, then the chromosome is *3 8 4 8 6 7*.

If the population size is 5, an initial population can be generated randomly as follows:

index	chromosome	fitness	
1	*3 8 4 8 6 7*	0.005	
2	7 3 7 6 1 3	0.062	
3	5 3 5 5 5 8	0.006	
4	7 6 7 7 2 2	0.020	
5	1 7 4 5 2 2	0.040	

2. Reproduction

There are two kinds of reproduction: generational reproduction and steady-state reproduction.

Generational Reproduction

In generational reproduction, the whole of a population is potentially replaced at each generation. The most often used procedure is to loop N/2 times, where N is the population size,

select *two* chromosomes each time according to the current selection procedure, producing *two* children from those two parents, finally producing N new chromosomes.

Steady-state Reproduction

The steady-state method selects two chromosomes according to the current selection procedure, performs crossover on them to obtain one or two children, perhaps applies mutation as well, and installs the result back into that population; the least fit of the population is destroyed.

3. Parent Selection mechanism

The effect of selection is to return a probabilistically selected parent. Although this selection procedure is stochastic, it does not imply GA employ a directionless search. The chance of each parent being selected is in some way related to its fitness.

Fitness-based selection

The standard, original method for parent selection is Roulette Wheel selection or fitness-based selection. In this kind of parent selection, each chromosome has a chance of selection that is directly proportional to its fitness. The effect of this depends on the range of fitness values in the current population.

Example: if fitness range from 5 to 10, then the fittest chromosome is twice as likely to be selected as a parent than the least fit.

If we apply fitness-based selection on the population given in example 3.1, we select the second chromosome 7 3 7 6 1 3 as our first parent and 1 7 4 5 2 2 as our second parent.

Rank-based selection

In the rank-based selection method, selection probabilities are based on a chromosome's relative rank or position in the population, rather than absolute fitness.

Tournament-based selection

The original tournament selection is to choose K parents at random and returns the fittest one of these.

4. Crossover Operator

The crossover operator is the most important in GA. Crossover is a process yielding recombination of bit strings via an exchange of segments between pairs of chromosomes. There are many kinds of crossover.

One-point Crossover

The procedure of one-point crossover is to randomly generate a number (less than or equal to the chromosome length) as the crossover position. Then, keep the bits before the number unchanged and swap the bits after the crossover position between the two parents.

Example: With the two parents selected above, we randomly generate a number 2 as the crossover position:

Parent1: 7 3 7 6 1 3

Parent2: 1 7 4 5 2 2

Then we get two children:

Child 1 : 7 3| 4 5 2 2

Child 2 : 1 7| 7 6 1 3

Two-point Cross Over

The procedure of two-point crossover is similar to that of one-point crossover except that we must select two positions and only the bits between the two positions are swapped. This crossover method can preserve the first and the last parts of a chromosome and just swap the middle part.

Example: With the two parents selected above, we randomly generate two numbers 2 and 4 as the crossover positions:

Parent1: 7 3 7 6 1 3

Parent2: 1 7 4 5 2 2

Then we get two children:

Child 1 : 7 3| 4 5| 1 3

Child 2 : 1 7| 7 6| 2 2

Uniform Crossover

The procedure of uniform crossover: each gene of the first parent has a 0.5 probability of swapping with the corresponding gene of the second parent.

Example: For each position, we randomly generate a number between 0 and 1, for example, 0.2, 0.7, 0.9, 0.4, 0.6, 0.1. If the number generated for a given position is less than 0.5, then child1 gets the gene from parent1, and child2 gets the gene from parent2. Otherwise, vice versa.

Parent1: 7 *3 *7 6 *1 3

Parent2: 1 *7 *4 5 *2 2

Then we get two children:

Child 1 : 7 7* 4* 6 2* 3

Child 2 : 1 3* 7* 5 1* 2

5. **Inversion**

Inversion operates as a kind of reordering technique. It operates on a single chromosome and inverts the order of the elements between two randomly chosen points on the chromosome. While this operator was inspired by a biological process, it requires additional overhead.

Example: Given a chromosome 3 8 4 8 6 7. If we randomly choose two positions 2, 5 and apply the inversion operator, then we get the new string: 3 6 8 4 8 7.

6. **Mutation**

Mutation has the effect of ensuring that all possible chromosomes are reachable. With crossover and even inversion, the search is constrained to alleles which exist in the initial population. The mutation operator can overcome this by simply randomly selecting any bit position in a string and changing it. This is useful since crossover and inversion may not be able to produce new alleles if they do not appear in the initial generation.

Example: Assume that we have already used crossover to get a new string: 7 3 4 5 1 3. Assume the mutation rate is 0.001 (usually a small value). Next, for the first bit 7, we generate randomly a number between 0 and 1. If the number is less than the mutation rate (0.001), then the first bit 7 needs to mutate. We generate another number between 1 and the maximum value 8, and get a number (for example 2). Now the first bit mutates to 2. We repeat the same procedure for the other bits. In our example, if only the first bit mutates, and the rest of the bits don't mutate, then we will get a new chromosome as below:

2 3 4 5 1 3

Applications of Genetic algorithms

GAs are good across a variety of problem domains. From optimization problems like:

- lens optimization
- prisoner's dilemma
- traveling salesman problem
- automated design - shape optimisation

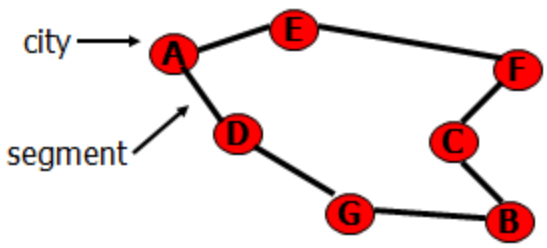
To other like:

- inductive concept learning
- layout planning problems
- routing problems
- control and system identification
- image processing
- marketing, credit and insurance modelling problems, stock prediction, credit scoring, risk assessment etc.

The results can be very good on some problems, and rather poor on others.

Example:

Traveling Salesman Problem



- objective: find the shortest tour that visits each city (NP-hard)
- a tour is encoded by a sequence AEFCBGD which specifies the order in which cities are visited
- fitness: overall length of the tour
- phenotype (tour) is defined by the permutation of genes rather than their values
- usual crossover results in invalid tours

  AEF|CBGD $\rightarrow$ AEF DFAB

  ECG|DFAB $\rightarrow$ ECG CBGD
- edge recombination operator

  parent 1 AEFCBDG

  parent 2 ECGDFAB

City	Connections
A	E,D,F,B
B	C,G,A,E
C	F,B,E,G
D	G,A,F
E	A,F,B,C
F	E,C,D,A
G	B,D,C

1. Select a start city
2. Select the neighbor with the minimum number of connections (break ties randomly)
3. Repeat 2 until no more cities are left

Tutorial 5:

What are the limitations of Genetic algorithms?
Explain constraint handling in genetic algorithms.